



Creating Graphical User Interface (GUI) Programs with wxGlade and wxGlue



GUI programming environments for Python

There are three packages for Python that support cross-platform GUI programming

- tkInter – an adaptation of the Tk GUI interface in Tcl/Tk. Produces Motif-like GUIs that look the same on all platforms. Not very attractive or versatile. Clumsy to program in Python.
- wxPython – a port of the wxWidgets multi-platform library. Produces native-looking GUIs on Windows/Linux/Mac. Many convenient GUI elements. Well documented. Well debugged. Complex to program.
- PyQT/PySide – a port of the QT multi-platform library. Produces native-looking GUIs on Windows/Linux/Mac. Well documented. Python bindings and distribution has been set back due to license conflicts. Likely to be better than wxPython soon.



wxPython GUI designer programs

- There are few programs that can be used to design GUI interfaces for wxPython
- Boa Constructor: powerful, confusing to use, produces messy code, no longer maintained
- wxGlade: limited, easier to use, produces clean code, active development
- wxFormBuilder: not tried
- DialogBlocks: commercial, looks interesting

All these programs help one design a GUI and provides a working script to produce it, but in the end one still needs to learn wxPython programming to integrate the GUI into a functioning application.



wxGlue

- Concept of wxGlue is to wrap objects around the wxPython code from a GUI builder that encapsulates all the wxPython code allowing one to use very simple code to develop fairly complex GUI programs.
 - One still needs to do some programming – but the task is much simpler
 - Developed using code developed in wxGlade, but in theory should work with code generated by other GUI development code
- Initial version completed 12/20/12
 - Supports all but ~6 of the controls (widgets) from the wxGlade palette.
- How to use:
 - Run wxGlade to create your GUI in a graphical editor; save as <MyGUI>.py file
 - Read the <MyGUI>.py file into wxGlue, a second <MyGUI>_shell.py is produced. This has example code for your GUI.
 - Edit the <MyGUI>_shell.py to utilize values from GUI controls or define actions to be done in response to them. All is done through simple “plain python” calls.



wxGlade: Where to find it and its documentation

- Home page: <http://wxglade.sourceforge.net/>
- Download with hg (mercurial) from: <https://bitbucket.org/agriggio/wxglade> (or zip file from <https://bitbucket.org/agriggio/wxglade/get/bfeac404735.zip>)
- Tutorials: <http://wxglade.sourceforge.net/tutorial.php> or <http://wiki.wxpython.org/WxGladeTutorial>



wxGlue: Where to find it and its documentation

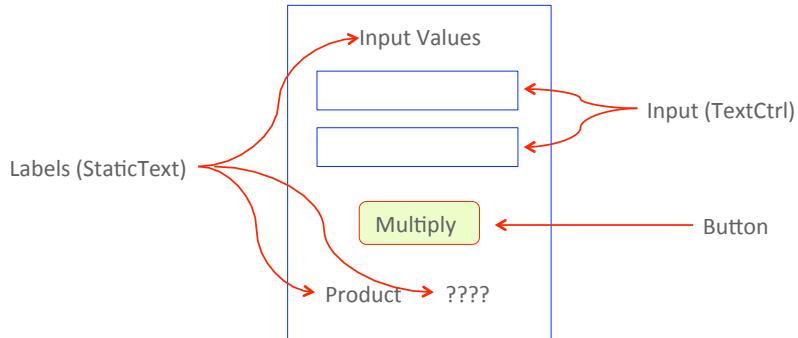
- Home page: (none at present)
- Download with subversion from:
<https://subversion.xray.aps.anl.gov/spec1ID/specpy/wxGlue/src/wxGlue>
- HTML documentation:
<https://subversion.xray.aps.anl.gov/spec1ID/specpy/trunk/docs/build/html/index.html>
- PDF documentation (48 pages, contents same as HTML):
<https://subversion.xray.aps.anl.gov/spec1ID/specpy/trunk/docs/build/latex/EPICSPythonSPECmotor.pdf>



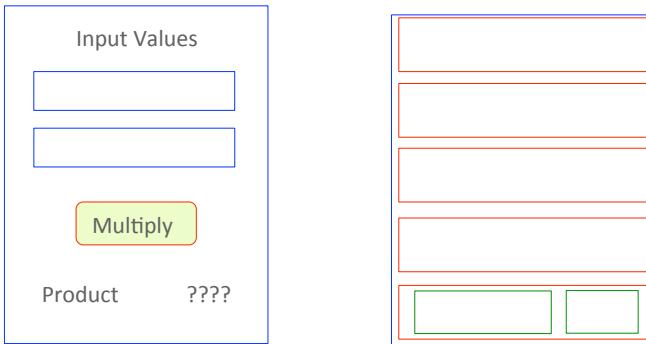
Let's design a short GUI application

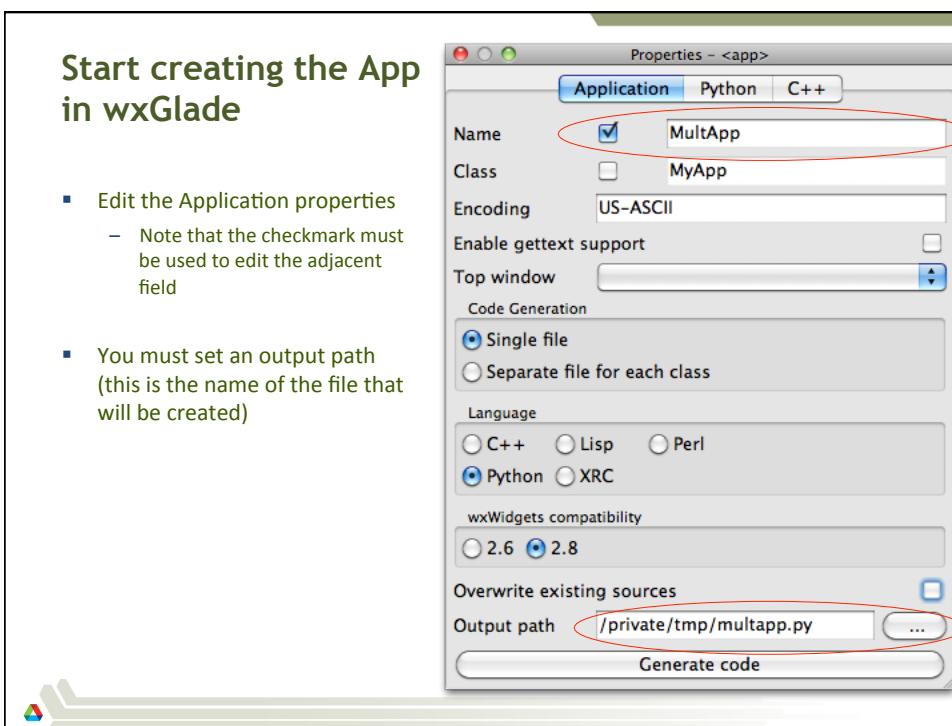
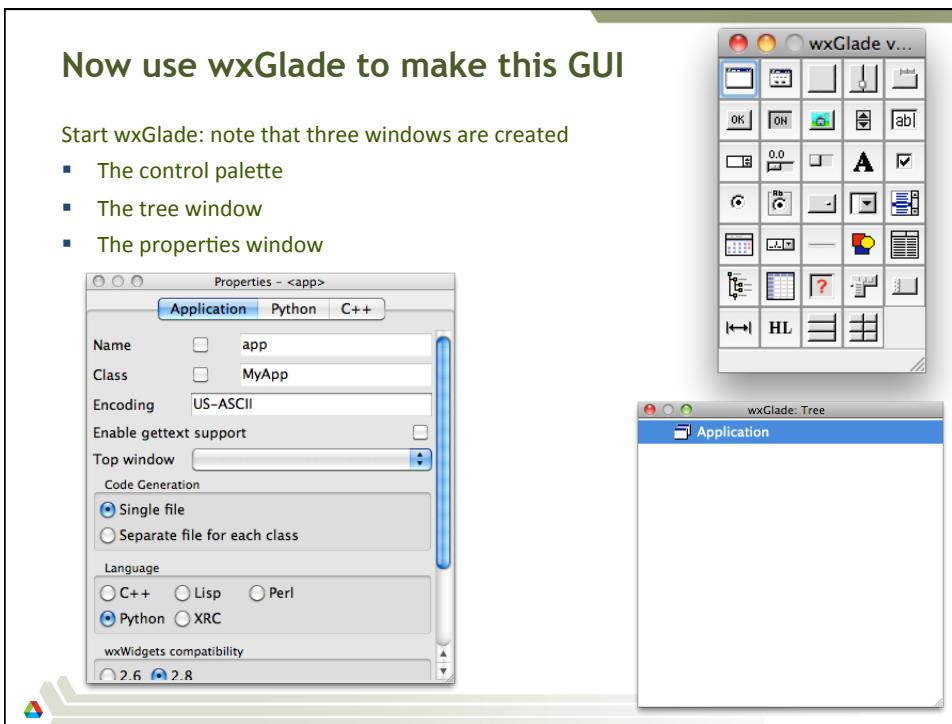
Goal: take two numbers as input and display the product when the compute button is pressed.

- Step 1: figure out what the GUI should look like and identify the types of controls that will be needed



- Step 2: figure out how to layout the GUI in terms of subdividing blocks (sizers)
 - Each block (sizer section) gets one control (widget)
 - Will need: A vertical sizer with five slots and where the last gets a horizontal size with two slots





Now add a frame to the app

- Click on the frame tool (a dialog would work for this, too.)
 - Note that wxPython uses the name “frame” for a window
- A window pops up with options for the new frame (defaults are fine – click OK).
- A frame and sizer get added to the application in the tree and a new design window is created.

The screenshot shows the wxGlade interface. On the right, there's a toolbar with various icons, one of which is circled in red. Below it is a 'Select fram...' dialog box with 'wxFrame' selected as the base class and 'MyFrame' entered in the Class field. To the right of the dialog is the 'wxGlade: Tree *' window, which displays a tree structure under the 'Application' node: 'frame_1 (MyFrame)' is expanded, showing 'sizer_1' as a child. On the left, there's a design window titled '<Design> - frame_1' showing a single rectangular panel with a diagonal hatching pattern.

Now set up main sizer

- Select the sizer in the tree (note properties window now has sizer info)
- Right-click on the sizer entry in the tree and select “Add slot”
 - Do this four times until there are five sections on the design window, as below

The screenshot shows the wxGlade interface. On the right, the 'wxGlade: Tree *' window has 'sizer_1' selected. A context menu is open with options: 'Remove', 'Add slot', 'Insert slot...', 'Copy' (with keyboard shortcut '⌘C'), 'Cut' (with keyboard shortcut '⌘X'), and 'Preview (frame_1)' (with keyboard shortcut '⌘P'). On the left, the design window titled '<Design> - frame_1' shows a vertical stack of five rectangular panels with a diagonal hatching pattern, representing the five sections of the sizer. At the bottom left, there's a small icon of a triangle pointing right.

Add a sizer to the bottom slot

- Click on the sizer entry in the palette. Move the mouse to the design frame.
 - Note the cursor changes to a plus sign.
 - Click in the bottom slot of the five Sizer_1 sections
- Note the select sizer type box appears. We want to add a horizontal sizer with two slots (change the slots value) and press OK.
- A new sizer appears in the tree
- The design window changes

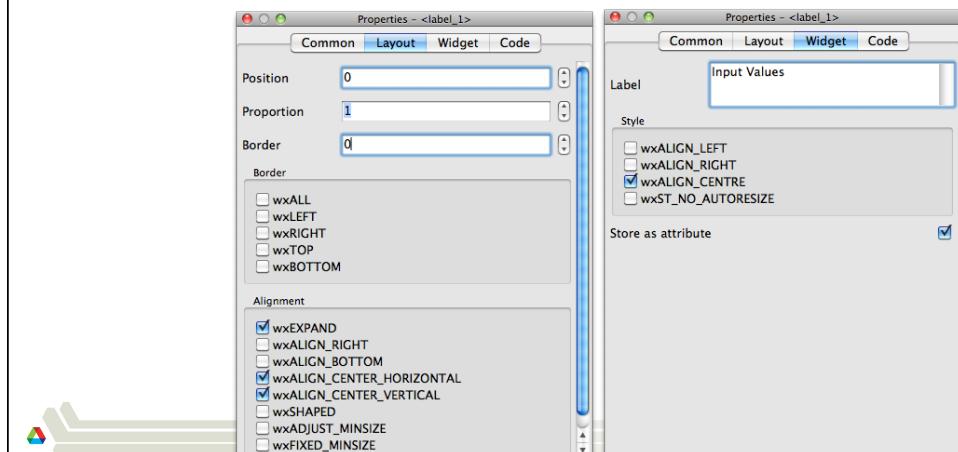
Now add controls to the sizer slots

Note that we will want labels (StaticText) in the three places marked below.

- Select the StaticText control. Click in the top slot.

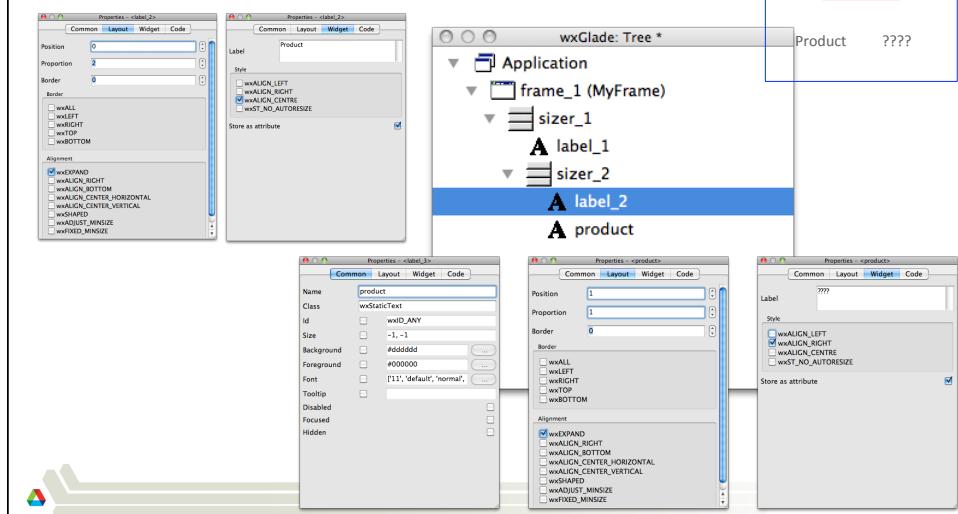
Now edit the Label's settings

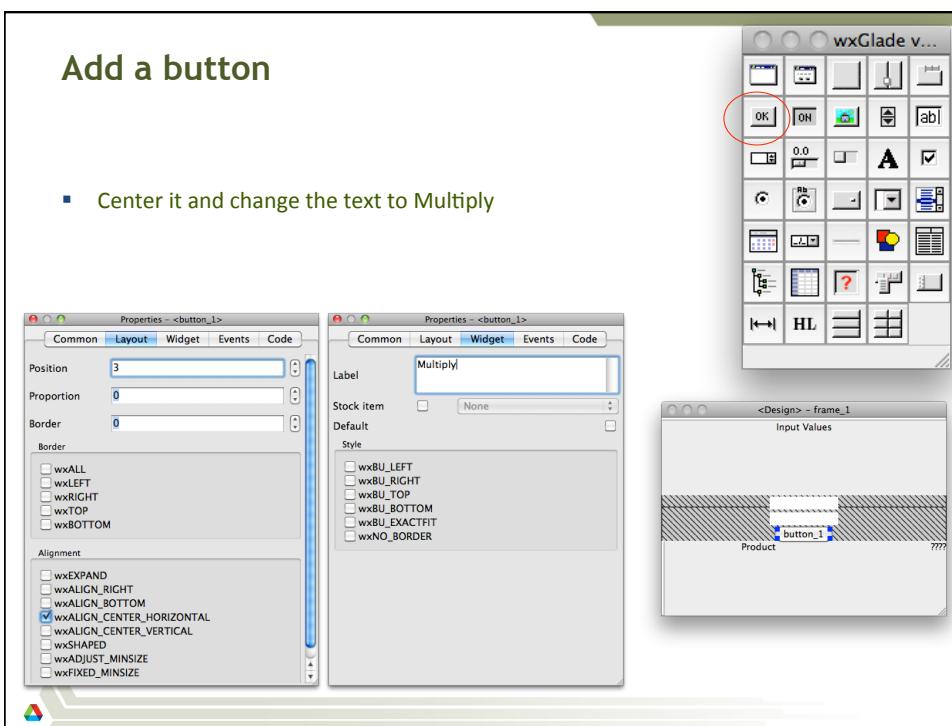
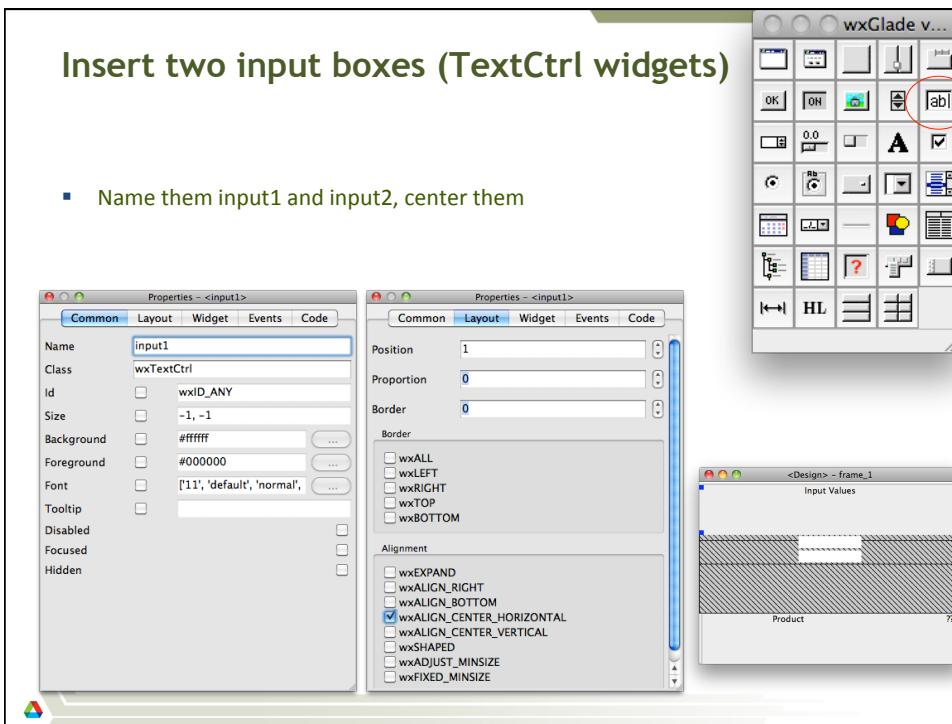
- Set both layout and widget properties



Repeat for other two labels

- For place where result goes, let's name that "product"





Generate the python code

- Use File/Generate Code
- Try running it


```
python /tmp/multapp.py
```

Looks OK, but would be better if more spacing left around controls (we will leave that as an exercise. Hint: use the border & alignment controls)

Look at the generated code

- Note that the multapp.py file contains ~60 lines of terse wxPython coding. Non-experts will not want to mess with that.

```

class MyFrame(wx.Frame):
    def __init__(self, *args, **kwdbs):
        # begin wxGlade: MyFrame.__init__
        kwdbs["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwdbs)
        self.label_1 = wx.StaticText(self, wx.ID_ANY, "Input Values", style=wx.ALIGN_CENTRE)
        self.input1 = wx.TextCtrl(self, wx.ID_ANY, "")
        self.input2 = wx.TextCtrl(self, wx.ID_ANY, "")
        self.button_1 = wx.Button(self, wx.ID_ANY, "Multiply")
        self.label_2 = wx.StaticText(self, wx.ID_ANY, "Product", style=wx.ALIGN_CENTRE)
        self.product = wx.StaticText(self, wx.ID_ANY, "????", style=wx.ALIGN_RIGHT)

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

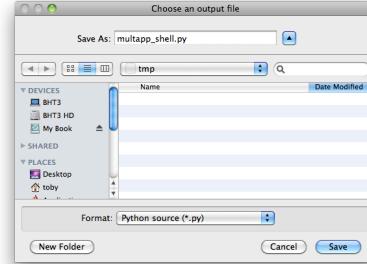
    def __set_properties(self):
        # begin wxGlade: MyFrame.__set_properties
        self.SetTitle("frame_1")
        self.SetSize((163, 224))
        # end wxGlade

    def __do_layout(self):

```

Now make the app do something!

- We have a nice GUI, but it does nothing useful. We need to “wire it up” to the code that actually does what we need. To do this “normally” we would need to write wxPython code to get values from the TextCtrl controls, respond to the button, etc.



Instead, this is where wxGlue can help.

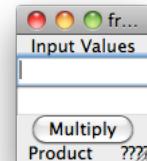
- Step 1 (to be streamlined later), give the multapp.py file to wxGlue to profile:

```
bht3:tmp toby$ python <path>wxGlue/_init__.py /tmp/multapp.py
```

Output from wxGlue

- wxGlue creates file multapp_shell.py
 - Running this gives us our same app
 - Something does happen if one clicks on the button, now
- The wxGlue output file contains ~50 lines of ordinary Python code (no wxPython). Lots are commented.

We will edit this code to make the app work.



```
python multapp_shell.py
```

```
toby@bht3:/tmp - bash - 80x24
import sys
sys.path.insert(0,"/Users/toby/software/wxGlue/src")
import wxGlue
sys.path.insert(0,"/tmp")
import multapp

def multapp_MyFrame(parent=None):
    gluemod = wxGlue.wxGlue(multapp.MyFrame,parent)
    def On_button_1():
        input2 = gluemod.GetValue('input2')
        input1 = gluemod.GetValue('input1')
        print('Called On_button_1 with values'
              +'\n\tinput2 = '+ str(input2)
              +'\n\tinput1 = '+ str(input1)
              )
    gluemod.SetAction("button_1", On_button_1)
    # gluemod.SetLabel("button_1", 'Button Label')
    # gluemod.Enable("button_1", True)
    # gluemod.SetValidationRequired("button_1", True)

    # gluemod.SetTextCtrlType("input1", float)
    # gluemod.SetValue("input1", 1.0)
```

Changes to make multapp_shell.py work (1)

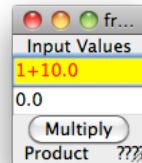
Step 1: Change the input controls to accept only floating point numbers and set the box to have an initial value (0.0). Do this for input1 and input2

Before:

```
# gluemod.SetTextCtrlType("input1", float)
# gluemod.SetValue("input1", 1.0)
```

After:

```
gluemod.SetTextCtrlType("input1", float)
gluemod.SetValue("input1", 0.0)
```



- If we run the app now, it will not accept non-digits (plus .+ eE). Invalid input causes the box to turn yellow.

Changes to make multapp_shell.py work (2)

Step 2: Change the button so it computes the sum.

- This line (already in file) causes the button to call function On_button_1:

```
gluemod.SetAction("button_1", On_button_1)
```

- This line (already in file) would set the contents of the "product" label

```
# gluemod.SetLabel("product", 'Control lbl')
```

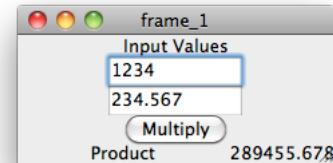
- This routine defines the default button response

```
def On_button_1():
    input2 = gluemod.GetValue('input2')
    input1 = gluemod.GetValue('input1')
    print('Called On_button_1 with values'
        +'\n\tinput2 = '+ str(input2)
        +'\n\tinput1 = '+ str(input1)
    )
```

Changes to make multapp_shell.py work (3)

- Change the routine to compute the sum and set the label. The last line resizes the window to fit all text

```
def On_button_1():
    input2 = gluemod.GetValue('input2')
    input1 = gluemod.GetValue('input1')
    prod = input1 * input2
    gluemod.SetLabel("product", str(prod))
    gluemod.ResizeFrame()
```



Changes to make multapp_shell.py work (4)

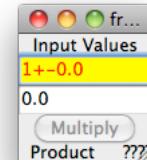
- Change the button usage, so that it is disabled when any inputs are yellow (invalid)

- Before

```
# gluemod.SetValidationRequired("button_1", True)
```

- After

```
gluemod.SetValidationRequired("button_1", True)
```



wxGlue coding

- Note that the completed wxGlue implementation has <25 lines of actual code (excluding comments and blank lines).
- This is all “plain” python
- Most of the “GUI” code follows examples in the template

```
import sys
sys.path.insert(0,"/Users/toby/software/wxGlue/src")
import wxGlue
sys.path.insert(0,"/tmp")
import multapp

def multapp_MyFrame(parent=None):
    gluemod = wxGlue.wxGlue(multapp.MyFrame,parent)
    def On_button_1():
        input2 = gluemod.GetValue('input2')
        input1 = gluemod.GetValue('input1')
        prod = input1 * input2
        gluemod.SetLabel("product", str(prod))
        gluemod.ResizeFrame()

    gluemod.setAction("button_1", On_button_1)
    gluemod.setValidationRequired("button_1", True)

    gluemod.SetTextCtrlType("input1", float)
    gluemod.SetValue("input1", 0.0)

    gluemod.SetTextCtrlType("input2", float)
    gluemod.SetValue("input2", 0.0)

    gluemod.ResizeFrame()
multapp_MyFrame()
wxGlue.StartEventLoop()
```

